CS61A Lecture #39: Conclusion

Announcements:

• HKN surveys TODAY: 5 bonus points for filling out their survey. Get your code from the sheets that we will circulate to put on your final for credit.

• We will also have an on-line survey (for EC). Watch website and Piazza.

• Last miniquiz to be released soon. Watch website, Piazza.

• A last (ungraded) homework will be released over the weekend: intended for review.

• Scheme Art Judging (see the website). Get your vote in (for credit) by next Wednesday.

• If you have regrade requests (or other grade issues), please get them to us by Sunday.

• Review sessions next week. Watch Piazza, website for announcements of topics and places.

• Otherwise, no standard office hours (except mine) next week.
A Summary of Topics

• Programming primitives
• Derived programming structures
• Programming-language concepts, design, and implementation
• Programming “Paradigms”
• Software engineering
• Analysis
• Side excursions
• What’s Next?
Programming Primitives

- Recursion: the all-encompassing repetitive construct; recursive thinking
- Pairs: A universal data-structuring tool.
- Functions as data values, functions on functions
- Exceptions: Dealing with errors.
- Classes.
Derived Programming Structures

- Can build almost anything from primitives.
- Although Python also has specialized implementations of some important data structures.

- Sequences:
  - Lists: traversals, searching, inserting, deleting (destructive and non-destructive)
  - Trees: traversals, binary search trees, constructing, inserting, deleting

- Maps.

- Sequences: creating, traversing, searching,

- Iterators, generators.

- Trees: uses, traversing, and searching.
Programming-Language Concepts, Design, Implementation

- Python was developed largely as a teaching language, and is simpler in many ways than other "production" languages...

- And yet, it is a good deal more powerful (as measured by work done per line of code) than these same languages.

- Still, as you've seen, there are problems, too: dynamic vs. static discovery of errors.

- Big item: scope (what instance of what definition applies to evaluation of an identifier). This is what environment diagrams are intended to model.
  - Alternative: dynamic scoping.

- Implementing a language [CS164]:
  - Interpreters
  - Trees as an intermediate language
  - Relationship of run-time environment representation to scope rules.
  - "Little" languages as a programming tool
Paradigms

• Functional programming: expressions, not statements; no side-effects; use of higher-order functions.

• Data-directed and object-oriented programming
  - Organize program around types of data, not functions
  - Inheritance
  - Interface vs. implementation

• Rule-based programming (Prolog)
  - Declarative rather than imperative
  - Rule → action idea.
  - Logic programming:
    * Pattern matching, pattern variables as a programming tool
    * Declarative and imperative interpretation
    * Application to parsing
Software Engineering

• Biggest ideas: Abstraction, separation of concerns
• Specification of a program vs. its implementation
  - Syntactic spec (header) vs. semantic spec (comment).
  - Example of multiple implementations for the same abstract behavior
• Testing: for every program, there is a test.
  - In “Extreme Programming” there is a test for every module.
• Software engineering implicit in all our software courses, explicit in CS169.
Analysis

- What we can measure when we measure speed:
  - Raw time.
  - Counts of selected representative operations.
  - Symbolic expressions of running time.
  - Best/worst case.

- Application of asymptotic notation ($\Theta(\cdot)$, etc.) to summarizing symbolic time measurements concisely.
Important Side Excursions

- Parallelism.

- Cryptography:
  - protecting integrity, privacy, and authenticity of data.
  - Symmetric (DES, Enigma) and asymmetric (public-key) methods.

- Computatability [CS172]: Some functions cannot be computed. Problems that are “near” such functions cannot be computed quickly.
What’s Next (Course-Wise)?

- **CS61B**: (conventional) data structures and languages
- **CS61C**: computing hardware as programmers see it.
- **CS170, CS172, CS174**: “Theory”—analysis and construction of algorithms, theoretical models of computation, use of probabilistic algorithms and analysis.
- **CS161**: Security
- **CS162**: Operating systems.
- **CS164**: Implementation of programming languages
- **CS160, CS169**: User interfaces, software engineering
- **CS188, CS189**: Artificial intelligence, Machine Learning
- **CS184**: Graphics
- **CS186**: Databases
What's Next (Course-Wise) (II)

- **CS191**: Quantum Computing.
- **CS C149**: Embedded Systems.
- **CS 150**: Digital Systems Design
- **CS194**: Special topics. (E.g.) computational biology; parallel software; data science; networks, crowds, and markets; cell phones as a computing platform.
- Plus graduate courses on these subjects and more.
- And please don't forget CS199 and research projects.
What's Next (Otherwise)?

• Programming contests.
• The open-source world: Go out and build something!
• And above all: Have Fun!